

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное  
учреждение  
высшего образования  
«Национальный исследовательский ядерный университет  
«МИФИ»

Лабораторная работа №1:  
«Оценка пропускной способности скрытых каналов»

По дисциплине «Способы построения скрытых каналов»

Выполнил студент группы Б19-515  
Щербакова Александра

Москва, 2023 г.

## № по списку группы - 18; $18 \bmod 13 + 1 = 6$ вариант задания

### Задание

- Модель штатной передачи информации: пакеты случайной длины передаются в случайные моменты времени
- Скрытый канал: модифицированный скрытый канал, основанный на изменении длин передаваемых пакетов
- Возможности закладки: модификация трафика, буферизация трафика

### Теория

Пусть  $S = s_0 \dots s_{k-1}$  — секретное сообщение, представляющее собой  $k$ -битную строку. Данная строка разбивается на подстроки перед отправкой.  $W_i = s_{iw} \dots s_{iw+w-1}$  —  $i$ -ая битовая подстрока строки  $S$ .  $SUM_i$  — десятичное представление  $W_i$ , вычисляемое по правилу:

$$SUM_i = \begin{cases} [W_i]_{10} - 2^{w-1}, i \bmod 2 = 0, \\ [W_i]_{10} - (2^{w-1} - 1), i \bmod 2 = 1. \end{cases} \quad (1)$$

Представим алгоритм передачи информации по скрытому каналу [18]:

- шаг 1 — А и В общаются в обычном режиме, записывают длины передаваемых пакетов в Справочник;
- шаг 2 — А и В случайным образом выбирают длину  $l$  из Справочника;
- шаг 3 — во время  $i$ -ой отправки сообщения А отправляет В сообщение длины  $l_{next} = l + SUM_i$ , Справочник обновляется добавлением в него  $l_{next}$ ;
- шаг 4 — В восстанавливает  $i$ -ое сообщение:  $SUM_i = (l_{next} - l)$  и вычисляет  $W_i$ ;
- шаг 5 — шаги два и три повторяются до тех пор, пока секретное сообщение не передастся до конца.

Замечания:

- А и В знают  $k, w, i$ ;
- если  $L_{max}$  — максимальная длина сообщений в начальном заполнении Справочника, то должно выполняться соотношение  $L_{max} > 2^w$ ;
- на третьем шаге, если  $l_{next} < 0$ , то  $l_{next} = l_{next} + L_{max}$ .

## Описание стенда



Реализованы 3 логических устройства на Python. Все три работают на localhost на следующих портах:

- 65010 - Прокси-сервер. Клиент-серверное взаимодействие реализовано с помощью сокетов. Играет роль устройства защиты, в которое внедрена закладка. В качестве параметра получает скрытое сообщение фиксированной длины.
- 65011 - Пользователь 1. Отправляет на прокси пакеты случайной длины в случайные моменты времени. Не подозревает о закладке.
- 65012 - Пользователь 2. Получает легитимную информацию от пользователя 1 с внедренной скрытой информацией. Здесь же реализован модуль, имитирующий работу злоумышленника (функция `decoder()`), который декодирует полученное скрытое сообщение.

В качестве средства анализа трафика используется Wireshark, слушающий localhost.

## Результаты работы

### Трафик в Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	2018	65011 → 65010 Len=1976
2	2.159169114	127.0.0.1	127.0.0.1	UDP	971	65010 → 65012 Len=929
3	2.655941758	127.0.0.1	127.0.0.1	UDP	1360	65011 → 65010 Len=1318
4	4.248277785	127.0.0.1	127.0.0.1	UDP	945	65010 → 65012 Len=903
5	4.702503243	127.0.0.1	127.0.0.1	UDP	1748	65011 → 65010 Len=1706
6	6.437767074	127.0.0.1	127.0.0.1	UDP	916	65010 → 65012 Len=874
7	7.524532695	127.0.0.1	127.0.0.1	UDP	1376	65011 → 65010 Len=1334
8	9.288706004	127.0.0.1	127.0.0.1	UDP	903	65010 → 65012 Len=861
9	10.169388101	127.0.0.1	127.0.0.1	UDP	1334	65011 → 65010 Len=1292
10	11.598953296	127.0.0.1	127.0.0.1	UDP	876	65010 → 65012 Len=834
11	12.834429395	127.0.0.1	127.0.0.1	UDP	1642	65011 → 65010 Len=1600
12	14.587927743	127.0.0.1	127.0.0.1	UDP	865	65010 → 65012 Len=823
13	14.877934571	127.0.0.1	127.0.0.1	UDP	2079	65011 → 65010 Len=2037
14	16.901972107	127.0.0.1	127.0.0.1	UDP	1311	65011 → 65010 Len=1269
15	17.552512660	127.0.0.1	127.0.0.1	UDP	769	65010 → 65012 Len=727
16	19.331934907	127.0.0.1	127.0.0.1	UDP	1077	65011 → 65010 Len=1035
17	19.668644931	127.0.0.1	127.0.0.1	UDP	747	65010 → 65012 Len=705
18	22.083785110	127.0.0.1	127.0.0.1	UDP	2043	65011 → 65010 Len=2001
19	22.385915228	127.0.0.1	127.0.0.1	UDP	729	65010 → 65012 Len=687

Исходное сообщение, отправленное по скрытому каналу:

```
client_sender.py      proxy.py      info.txt
1 | Secret information transferring via covert channel len=64 bit!!
```

Отправляется только 64 бита (8 символов) = “Secret i”

Результат декодирования сообщения:

```
aleksandrishche@vb:~/fin$ sudo python3 client_listener.py
[sudo] password for aleksandrishche:
Client 2 has started
929
01010011
903
0101001101100101
874
010100110110010101100011
861
01010011011001010110001101110010
834
0101001101100101011000110111001001100101
823
010100110110010101100011011100100110010101110100
727
01010011011001010110001101110010011001010111010000100000
705
0101001101100101011000110111001001100101011101000010000001101001
Bits of msg: 0101001101100101011000110111001001100101011101000010000001101001
Decoded msg: Secret i
```

## Заключение

В ходе данной лабораторной работы были получены практические навыки создания программных модулей для построения скрытых каналов, а также изучены принципы построения скрытых каналов по памяти.

## Приложение

*Файл config.py*

```
SECOND_CLIENT_ADDRESS = ('127.0.0.1', 65012)
FIRST_CLIENT_ADDRESS = ('127.0.0.1', 65011)
PROXY_ADDRESS = ('127.0.0.1', 65010)

K = 64
W = 8
```

*Файл info.txt*

```
Secret information transferring via covert channel len=64 bit!!
```

*Файл proxy.py*

```
import socket
import threading
import argparse
import queue
from scapy.all import *
from scapy.layers.inet import *
from config import *
import string

soc = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
soc.bind(PROXY_ADDRESS)

random.seed(1000000)

# Имитация сетевого экрана
def listener():
    soc = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```

soc.bind(PROXY_ADDRESS)
while True:
    data, address = soc.recvfrom(1024)
    if address[1] == 65011:
        soc.sendto(data, SECOND_CLIENT_ADDRESS)
    if address[1] == 65012:
        soc.sendto(data, FIRST_CLIENT_ADDRESS)

class MyConcurrentCollection:
    def __init__(self):
        self.collection = queue.Queue()

    def append(self, x):
        self.collection.put(x)

    def pop(self):
        return self.collection.get()

    def __len__(self):
        return self.collection.qsize()

    def __str__(self):
        return f"{len(self)}"

    def print_collection(self):
        return self.collection.queue

    def empty(self):
        return self.collection.empty()

class Worker(threading.Thread):
    def __init__(self, msg, input_collection:
MyConcurrentCollection, callback=None):
        if not callback:
            raise NameError('callback not set')
        threading.Thread.__init__(self)
        self.daemon = True
        self.msg = ''.join(format(ord(x), '08b') for x in msg)
        self.input_collection = input_collection
        self.callback = callback

```

```

        self.i = 0

    def run(self):
        print(self.msg)
        while True:
            if len(self.msg) == 0:
                print("Covert channel close, all message send")
                return
            if not self.input_collection.empty():
                print(self.i // W)
                packet = self.input_collection.pop()
                if self.i == 0:
                    with open('Dictionary.txt', 'w') as f:
                        pass
                    covert_message = self.msg[:W]
                    self.msg = self.msg[W:]
                    self.callback(covert_message, packet, self.i)
                    self.i += W
                else:
                    time.sleep(0.1)

class Agent:
    def __init__(self, msg, callback=None, threads_count = 1):
        self.col = MyConcurrentCollection()
        self.consumers = [Worker(msg, self.col, callback) for _ in
range(threads_count)]

    def sniffer(self):
        sniff(filter="src port 65011 and dst port 65010",
prn=self.col.append, iface="lo")

    def run(self):
        for consumer in self.consumers:
            consumer.start()
        self.sniffer()
        for consumer in self.consumers:
            consumer.join()

```

```

def calc_sum_i(cur_i, w_i):
    sum_i = 0
    if cur_i % 2 == 0:
        sum_i += int(w_i, 2) - 2 ** (W - 1)
    else:
        sum_i += int(w_i, 2) - (2 ** (W - 1) - 1)
    return sum_i

def covert(covert_message, pkt, i):
    if i == 0:
        with open('Dictionary.txt', 'a+') as f:
            l_max = random.randrange(2 ** W + 1, 3 ** W)
            cur_l = l_max
            l_next = cur_l
            f.write(str(l_max) + '\n')
    else:
        with open('Dictionary.txt', 'r') as f:
            all_ls = f.read().split('\n')[:-1]
            l_max = int(all_ls[0])
            l_next = int(all_ls[len(all_ls) - 1])
            cur_l = l_next

    cur_l = l_next
    sum_i = Agent.calc_sum_i(i // W, covert_message)
    l_next = sum_i + cur_l if sum_i + cur_l > 0 else sum_i +
cur_l + l_max

    with open('Dictionary.txt', 'a') as f:
        f.write(str(l_next) + '\n')
    msg_to_send = ''.join(random.choices(string.ascii_letters,
k=l_next))
    l_max = cur_l
    time.sleep(random.uniform(2,3))
    soc.sendto(msg_to_send.encode(), SECOND_CLIENT_ADDRESS)
    print(len(msg_to_send))

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Covert channel
emulation')

```



```

    parser.add_argument('-f', '--filename', help='data to transfer
via covert channel', required=False, dest='filename', type=str)
    args = parser.parse_args()
    if args.filename:
        file = open(args.filename)
        msg = file.read()
    else:
        msg = input()

    agent = Agent(msg, Agent.covert)
    thread_agent = threading.Thread(target=agent.run, args=())
    #thread_proxy = threading.Thread(target=listener, args=())

    thread_agent.start()
    #thread_proxy.start()

    thread_agent.join()
    #thread_proxy.join()
    #sniff()

```

### *Файл client\_sender.py*

```

import random
import socket
import time
from random import randint
from config import *

print("Client 1 has started")
soc = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
soc.bind(FIRST_CLIENT_ADDRESS)

def sender():
    while True:
        try:
            time_parameter = random.uniform(2, 3)
            buffer = random.randbytes(randint(1024, 2048)) # по
условию передаются пакеты случайной длины

```

```

        soc.sendto(buffer, PROXY_ADDRESS)
        time.sleep(time_parameter)    # по условию пакеты
передаются в случайные моменты времени
    except (KeyboardInterrupt, EOFError) as e:
        soc.close()
        break;

if __name__ == '__main__':
    sender()

```

*Файл client\_listener.py*

```

from scapy.all import *
import socket
from config import *
# client outside the IS who recieves the legal info as well as
secrete message that was added by proxy

print("Client 2 has started")
random.seed(1000000)
soc = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # udp -
"socket.SOCK_DGRAM", tcp - Stream
soc.bind(SECOND_CLIENT_ADDRESS)
soc.connect(PROXY_ADDRESS)

def listener():
    cur_i = 0
    bits_of_msg = ''
    while True:
        data = soc.recv(10240)
        #pac = sniff(filter="src port 65012 and dst port 65011")
        #print(pac)
        print(len(data))
        bits_of_msg += decoder(data, cur_i)
        print(bits_of_msg)
        cur_i += 1
        if cur_i == K // W:

```

```

        print('Bits of msg:', bits_of_msg)
        print('Decoded msg:',
''.join([chr(int(bits_of_msg[i:i+8], 2)) for i in range(0,
len(bits_of_msg), 8)]))
        break

def decoder(msg, cur_i):
    l_next = len(msg)
    l_max = 0
    all_ls = []
    with open('Dictionary.txt', 'r') as f:
        all_ls = f.read().split('\n')
        l_max = int(all_ls[cur_i])

    sum_i = l_next - l_max
    w_i = sum_i
    if cur_i % 2 == 0:
        w_i += 2 ** (W - 1)
    else:
        w_i += (2 ** (W - 1) - 1)
    return format(w_i, f'0{W}b')

if __name__ == '__main__':
    listener()

```